Was wir heute tun

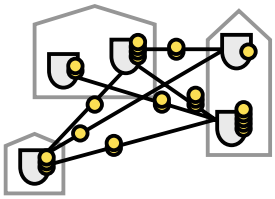
5. Juni 2003

- Die Welt der Banken 
- Dijkstra & seine Philosophen 
- vom Rendezvous des schlafenden Coiffeurs 
- Sieb des Erathostenes 
- Reader & Writer 

Die Welt der Banken

- besteht aus Geld & Besitzern (Konto)
- Geld kann Besitzer wechseln (Transaktion)
- es gibt immer gleich viel Geld (Invariante)

Banken sind Stellvertreter für mehrere Besitzer
Banken übernehmen Kontrolle für Invariante



Was kann ein Konto?

```
interface Konto {
  void abheben( Betrag );
  void einzahlen( Betrag );
  Betrag Saldo;
}
```

Was ist eine Transaktion?

```
void Transaktion( von, nach, Betrag )
{
    // überweise
    von.abheben( Betrag );
    nach.einzahlen( Betrag );
}
```

Invariante „es gibt immer gleich viel Geld“ soll erhalten bleiben

Immer gleich viel Geld

was passiert, wenn zwei Threads gleichzeitig
- von A abheben?
- auf B einzahlen?

```
void Transaktion( von, nach, Betrag )
{
    // blockiere von bzw. nach
    // geeignet

    von.abheben( Betrag );

    nach.einzahlen( Betrag );

}
```

Totale Blockierung

```
void Transaktion( von, nach, Betrag )
{
    lock( von ); lock( nach );

    von.abheben( Betrag );

    nach.einzahlen( Betrag );

}
```

was kann passieren, wenn Folgendes ausgeführt wird:

```
Transaktion( A, B, 13 );
Transaktion( B, A, 43 );
```

Aufsteigend blockieren

Kardinalität der Konti (A < B < ... < ZZZ)

```
void Transaktion( von, nach, Betrag )
{
     if ( von < nach ) {
          lock( von ); lock( nach );
     } else {
          lock( nach ); lock( von );

     von.abheben( Betrag );

     nach.einzahlen( Betrag );

}
```

Separates Blockieren

immer nur, was benutzt wird blockieren:

```
void Transaktion( von, nach, Betrag )
{
     lock( von );
     von.abheben( Betrag );

     lock( nach );
     nach.einzahlen( Betrag );

}
```

was kann hier passieren?

Alle Konti summieren

was muss der Revisor tun?

```
int Revisor() {

     foreach( Konto k in konti )
          s += k.Saldo;

     return s;

}
```

Transaktionen finden dazwischen statt...

Revisors Blockierungen

Transaktionen blockieren „aufsteigend“

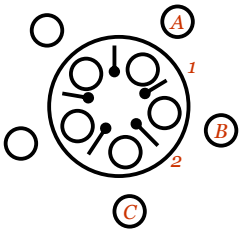
```
int Revisor() {

     // alles sperren
     foreach( Konto k in konti )
          Monitor.Enter( k );

     for each( Konto k in konti ) {
          s += k.Saldo;
          Monitor.Exit( k );
     }

}
```

Dijkstra & seine Philosophen

fünf Philosophen sitzen um einen Tisch

jeder Philosoph denkt oder isst

ein Philosoph isst nur mit zwei Gabeln

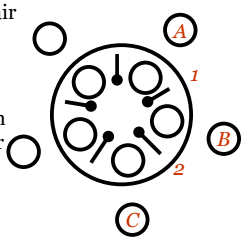


Schicksale für Philosophen

ein Philosoph, der hungrig ist, muss so lange warten, bis beide Gabel links und rechts frei sind

es kann passieren, dass ein hungriger Philosoph nie zum Essen kommt
→ er verhungert
→ die Verteilung ist unfair

es kann passieren, dass alle gleichzeitig hungrig sind, die linke Gabel nehmen und ewig warten
→ es wird nie mehr einer essen
→ Situation verklemmt



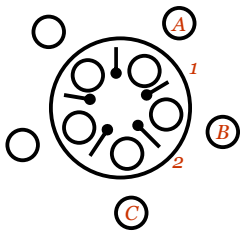
Begriffe

Synchronisation: Koordination nebenläufiger Prozesse auf gemeinsame Ressource

Deadlock: kein Prozess kann fortfahren, bevor ein anderer ebenfalls wartender Prozess fortfährt

Starvation: ein Prozess wird nie zur Ausführung zugelassen, obwohl er noch nicht beendet ist

Fairness: jedem wartenden Prozess wird irgendwann eine benötigte Ressource zugeteilt.



Erathostenes von Kyrene

284 - 202 v.Chr.

Mathematiker, Astronom, Geograph

Freund des Archimedes

bestimmte erstmals Umfang der Erde mit Hilfe des Zenitwinkels

Das Sieb des Erathostenes

Zuerst markiert man alle Zahlen im Bereich von 2 bis x als Primzahlen. Dann beginnt man mit der 2 und entfernt bei allen Vielfachen von 2, d.h. 4, 6, 8, ... , die Markierung. Wenn man die Obergrenze x erreicht oder überschritten hat, ist man mit der ersten Runde fertig. Dann sucht man von vorn beginnend die nächste Zahl, die markiert ist und demarkiert alle ihrer Vielfachen. Damit fährt man fort, bis man die Vielfachen aller Zahlen im Bereich 2 bis x demarkiert hat. Alle Zahlen, die danach noch markiert sind, sind Primzahlen.

Das Sleeping Barber Problem